Machine Learning

Coursera - Stanford University

Andrew Ng

Summarised By:

Sanjay Singh

san.singhsanjay@gmail.com

Week 1

Introduction

- Machine Learning is the science of getting computers to act without being explicitly programmed.
- You use ML dozens of times a day:
  - When you do a Google search
  - When Facebooks tags you in a picture
  - Your spam filter
  - Etc.
- Machine Learning:
  - Grew out of work in AI: we wanted our machines to be as much intelligent as human. To do this, we realised the only way is ML.
  - New capability for computers: To make our computers better than earlier.
- Some examples of applications of ML:
  - Database mining: Large database from growth of automation / web.
    - Web click data
    - Medical Records
    - Biology – like genome data set
    - Engineering – almost every field
  - Applications cannot programmed by hand:
    - Autonomous helicopter
    - Handwritten recognition
    - Natural Language Processing (NLP)
    - Computer Vision
  - Self customization programs: Amazon and Netflix product recommendations.
  - Understanding human learning (brain, real AI).

Introduction

- There is no universally accepted definition of Machine Learning (ML).
- Following definition is an old one given by Arthur Samuel in 1959:

  "Machine Learning is the field of study that gives computers the ability to learn with out being explicitly programmed."
- The most recent definition of Machine Learning (ML) was given by Tom Mithcell in 1998:

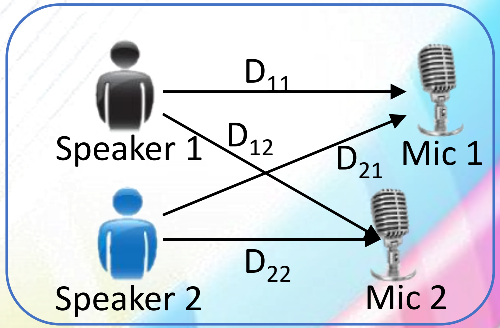
  "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."
- Mainly, there are two types of Machine Learning algorithms:
  1. Supervised Learning:

     Right answers are provided with each instance of data set for the learning of algorithm.
  2. Unsupervised Learning:

     Basically, it is a clustering technique. This is used to find out different unknown types or classes.
- Other ML algorithms are:
  1. Reinforcement Learning
  2. Recommender Systems

Introduction

- Cocktail Party problem can be solved by using Unsupervised Learning problem.
- Following is the Cocktail Party problem:
  - There are n number of speakers speaking simultaneously (suppose n = 2).
  - There are n number of mics (suppose n = 2).
  - These mics are at different distance from each speaker.
  - Each mic is comparatively closer to one speaker and farther from others.
  - Each of these mics will record the composed voice / signal of every speaker.
  - Problem is to separate out the voice (or signal) of each speaker.



- Above problem is one the most complicated problem in the field of Computer Science and Signal Processing. However, it can be solved with the help of Unsupervised Learning Algorithms.
- Following is a one line unsupervised solution for Cocktail Party problem:

*[W, s, v] = svd((repmat(sum(x.*x, 1), size(x, 1), 1). *x)*x');*

[Source: Sam Roweis, Yair Weiss & Eero Simoncelli]

Introduction

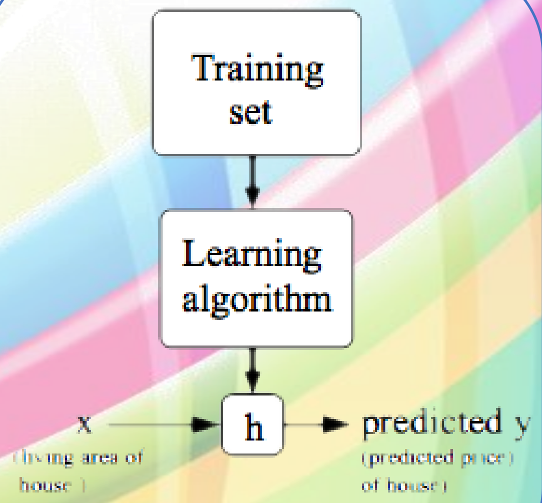
- We will use GNU Octave to implement these ML algorithms due to following reasons:
  - Octave is free and an open source alternative of MATLAB.
  - Octave has a good collection of computation and numerical libraries.
  - All the complicated mathematical functions are available in Octave, hence it would be quite fast and easy to implement them in Octave.
  - Doing the similar thing in any other language, like Python, C / C++, Java, etc., will take a long time and efforts.
- Many people in Silicon Valley use Octave to prototype their Machine Learning algorithms as it is easy and quick. Only after getting a desired result / output, then start implementing it in the required programming language, like C / C++, Java, Python, etc.

Model and Cost Function

Notations

- Following are some notations used in the further lectures:

Notation	Explanation
m	Number of training examples or instances
x	Input variable or feature
y	Output or target variable or feature
(x, y)	One training instance or example (any)
(x^i, y^i)	i^{th} training instance or example
h	Hypothesis: the output function of a learning algorithm mapping input and output



Model and Cost Function

- The hypothesis function of machine learning model looks something like this:

$$h(x) = \theta_0 + \theta_1 x$$

Where,

θ_i: Parameter

- Cost function: Find the value of θ_0 and θ_1 such that the half of average of sum of square of difference of predicted and actual value can be minimised:

$$J(\theta_0, \theta_1) = \min_{(\theta_0,\theta_1)} \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^i - yi)^2 = \min_{(\theta_0,\theta_1)} \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - yi)^2$$

Where,

m: No. of training examples	h: hypothesis function	$h(x^i)$: prediction for x^i
θ_i: i^{th} parameter	y^i: actual value for x^i	$h(x^i) = \theta_0 + \theta_1 x^i$
1/(2m): Avg. & simplify the math	$\hat{y}^i$: predicted value	

- $J(\theta_0, \theta_1)$ is a cost function, known as Squared Error Function or Mean Squared Error (MSE). It is one of the most commonly used cost function for regression problems.
- The mean is halved as a convenience for the computation of the gradient descent as the derivative term of the square function will cancel out the $\frac{1}{2}$ term.

Week 1

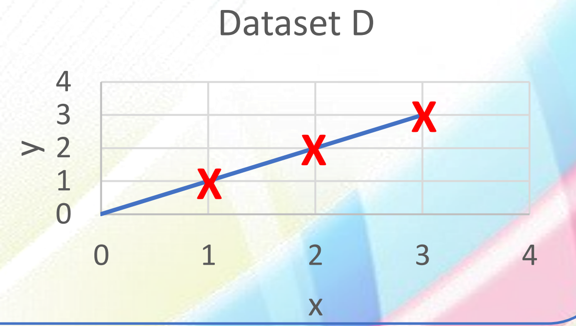
Model and Cost Function

Cost Function Intuition – I

- We have a dataset D with total number of instances, m = 3

Dataset D

Sr. No.	**x**	**y**
1	1	1
2	2	2
3	3	3



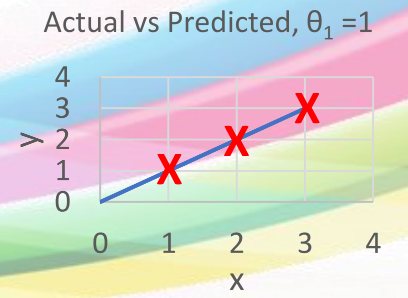
- Let's assume that the hypothesis function is: h(x) = θ_1 x. Thus,

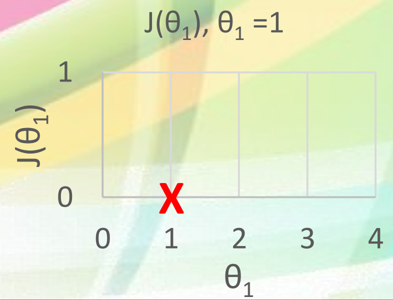
$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h(xi) - y^i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(\theta_1\,\text{xi} - yi)^2$$

For θ_1 = 1,

h(1) = 1, h(2) = 2, h(3) = 3

J(1) = $\frac{1}{6}$ [(1 - 1)^2 + (2 - 2)^2 + (3 – 3)^2] = 0





Week 1

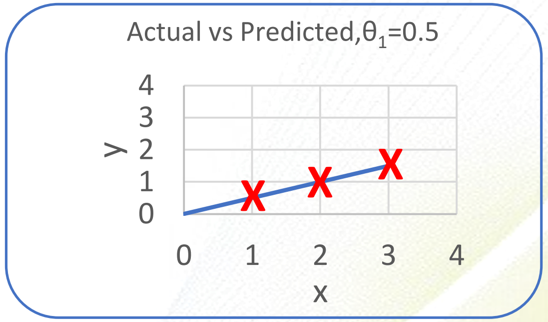
Model and Cost Function

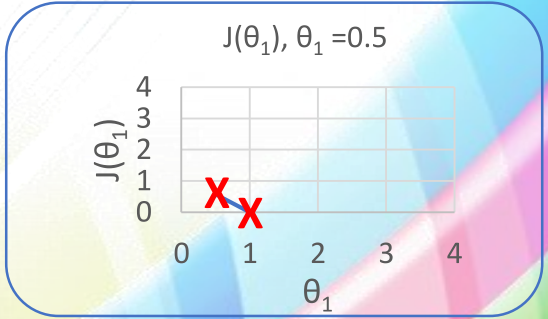
Cost Function Intuition – I

For $\theta_1 = 0.5$,

$h(1) = 0.5$, $h(2) = 1$, $h(3) = 1.5$

$$J(1) = \frac{1}{6}\left[(0.5 - 1)^2 + (1 - 2)^2 + (1.5 - 3)^2\right] = 0.58$$

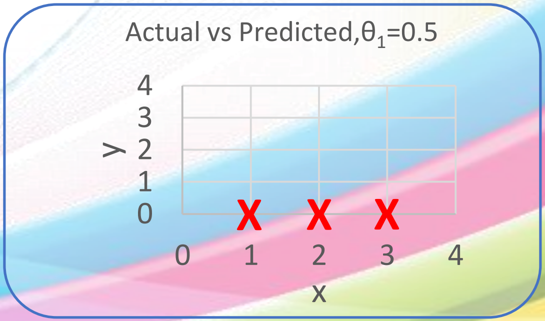


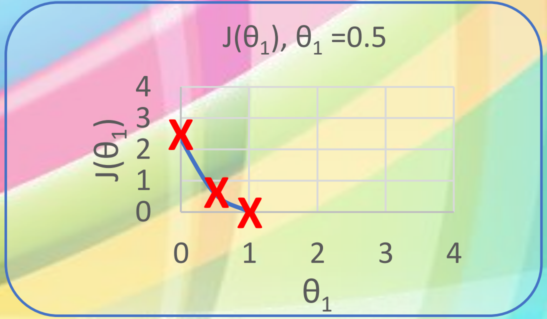


For $\theta_1 = 0$,

$h(1) = 0$, $h(2) = 0$, $h(3) = 0$

$$J(1) = \frac{1}{6}\left[(0 - 1)^2 + (0 - 2)^2 + (0 - 3)^2\right] = 2.33$$

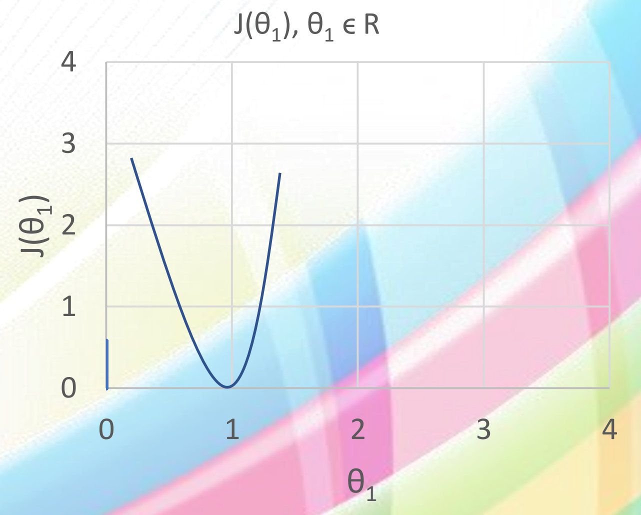




Model and Cost Function

Cost Function Intuition – I

- Hence, when we play more with the value of θ_1, we will get a plot something like depicted next to this.
- This plot shows that for every cost function $J(\theta_1)$, there is a value of θ_1 at which the value of $J(\theta_1)$, i.e., cost, becomes minimum or zero.
- Thus, we should to minimize the cost function. In this case, θ_1 = 1, is our global minimum.



Model and Cost Function

Cost Function – Intuition II

- In case of cost function, what we know so far is following:

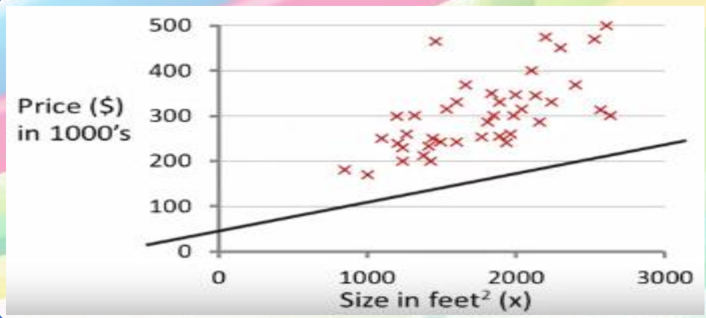
Hypothesis: $h(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h(xi) - y^i)^2$

Goal: $\min_{(\theta_0, \theta_1)} J(\theta_0, \theta_1)$

- Unlike "Cost Function – Intuition I" (last slides), this time we will use both the parameters, θ_0 & θ_1.
- Suppose we have a housing price data set and the value of our parameters is $\theta_0 = 50$ & $\theta_1 = 0.06$. Then, the plot of our data set looks something like this:

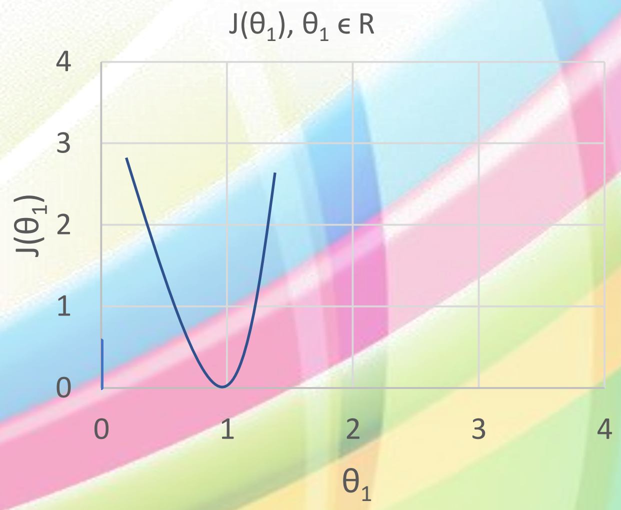


Week 1

Model and Cost Function

Cost Function – Intuition II

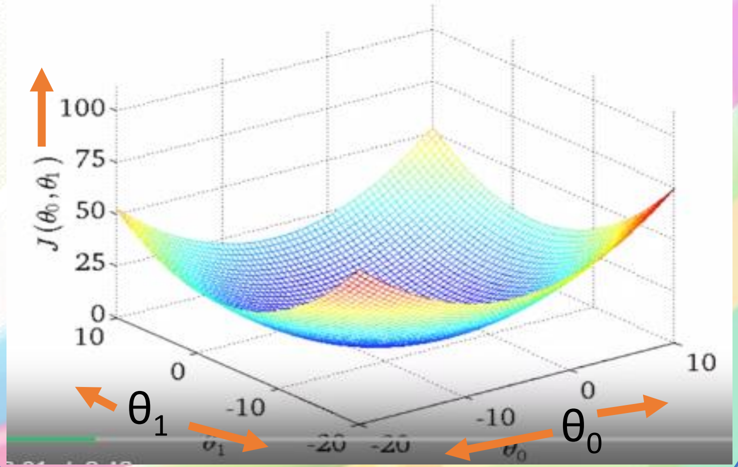
When we had only one parameter, θ_1 in "Cost Function – Intuition I", the plot of our cost function was like this (bowl shaped):



Model and Cost Function

Cost Function – Intuition II

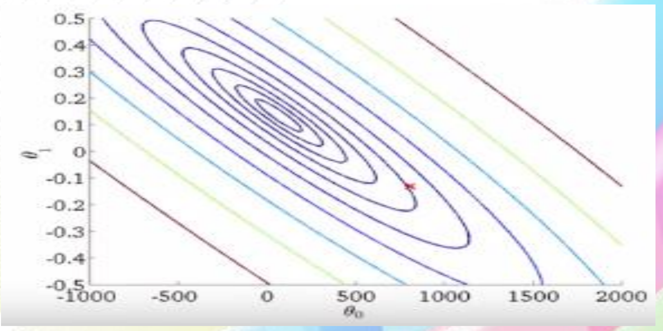
In case of two parameters, θ_0 & θ_1, the shape of cost function will become something like this (bowl shaped surface) depending on your dataset:

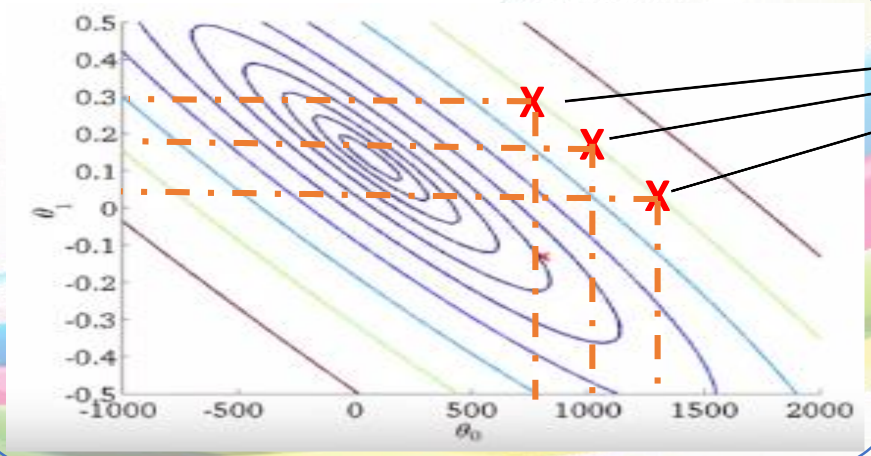


Model and Cost Function

Cost Function – Intuition II

- Instead of making 3D plots of parameters and cost function, like earlier, we will mostly create Contour Plots or Contour Figures.
- In the following plot, θ_0 is on x-axis & θ_1 is on y-axis.
- Each ellipse inside the plot is showing that $J(\theta_0, \theta_1)$ is getting same value as the height of ellipse is same from the two axes.
- Centre of all concentric ellipse is the minimum of $J(\theta_0, \theta_1)$.
- For example:



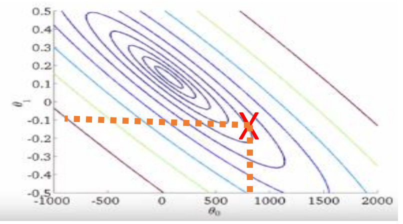


All these three points have the same value for $J(\theta_0, \theta_1)$ as they are on the same ellipse, but the values of θ_0 & θ_1 are different for all three.

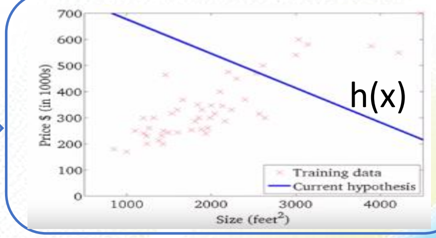
Model and Cost Function

Cost Function – Intuition II

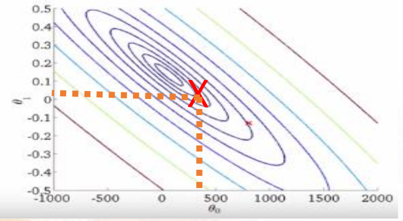
- Let us consider different values for θ_0 & θ_1 on the contour and see the plot of corresponding hypothesis (size of flat on x-axis and price is on y-axis, for every value of θ_0 & θ_1 :



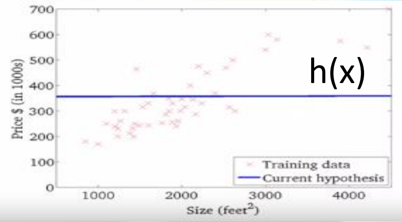
$\theta_0 = 800, \theta_1 = -0.1$
$y^i = h(x^i) = \theta_0 + \theta_1 x^i$
$y^i = h(x^i) = 800 - 0.1x^i$



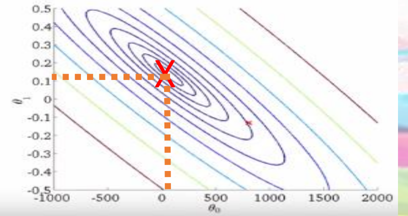
Here, the plot of h(x) is extremely inaccurate due to unoptimized values of θ_0 & θ_1.



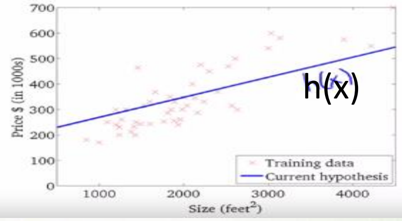
$\theta_0 = 360, \theta_1 = 0$
$y^i = h(x^i) = \theta_0 + \theta_1 x^i$
$y^i = h(x^i) = 360 + 0x^i$



Since θ_1 is zero here, hence for any value of x, h(x) will always be θ_0, that is 360.



$\theta_0 = 250, \theta_1 = 0.15$
$y^i = h(x^i) = \theta_0 + \theta_1 x^i$
$y^i = h(x^i) = 250 + 0.15x^i$



Here, the line of h(x) is quite accurate. Thus, the error is very less for these values of θ_0 & θ_1.

Week 1

Parameter Learning

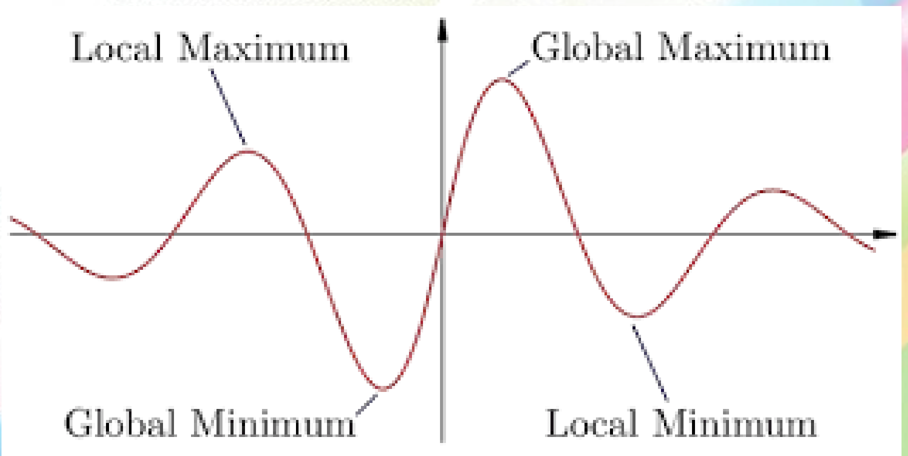
Gradient Descent: Prerequisite

Local Minimum / Maximum

- To get a local minimum or maximum, we need to have an interval.
- The local minimum or maximum is the point in the plot of a function which is at the lowest / highest position within that interval.

Global / Absolute Minimum / Maximum

- Global minimum or maximum is also called as absolute minimum or maximum
- This is the point in the entire plot of a function which is at the highest or lowest position in the entire plot.
- A global min. / max. can be only one.



Parameter Learning

Gradient Descent

- Gradient Descent is an algorithm to minimize the cost function $J(\theta_0, \theta_1)$.
- Gradient Descent is used everywhere in Machine Learning.
- First, we will use Gradient Descent to minimize some random function and later, we will use it to minimize some cost function.
- Following are the background and outline of our objective:

  Some function: $J(\theta_0, \theta_1)$

  Objective: $\min_{(\theta_0,\theta_1)} J(\theta_0, \theta_1)$

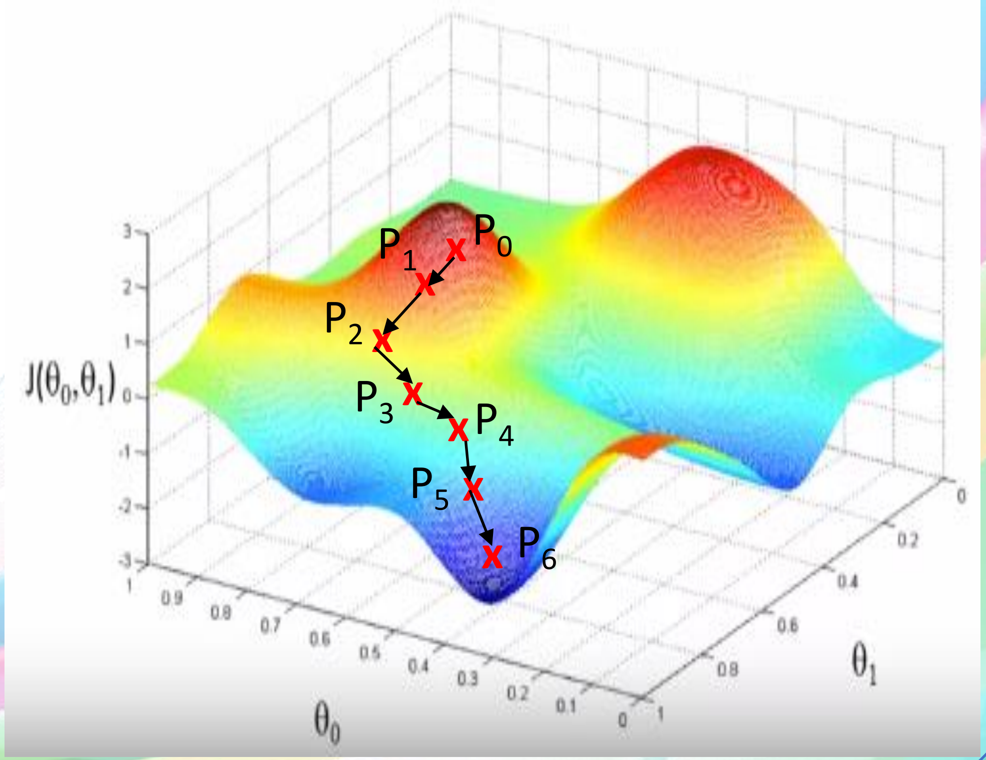
  Outline:
  - Start with some θ_0 & θ_1
  - Keep changing θ_0 & θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum
- Gradient Descent can be applied on general functions like: $J(\theta_0, \theta_1, \theta_2, \ldots\ldots, \theta_n)$
- To understand this, we will consider only two parameters: θ_0 & θ_1.
- A common choice to initialize the value of parameters is 0, i.e., $\theta_0 = 0$ & $\theta_1 = 0$.
- There are chances that by applying Gradient Descent we may end up at a local minimum.

Parameter Learning

Gradient Descent

What Gradient Descent does?

- Suppose you have initialised θ_0 & θ_1 with some random value (that can be 0, as well), due to which you are at position P_0 in the shown graph.
- Now, Gradient Descent will help us to find out the direction in which the value of our cost function $J(\theta_0, \theta_1)$ will reduce most by moving a tiny step (or baby step ☺) in that direction.
- Consequently, we will reach at P_1. Again, Gradient Descent will help us to find a direction when the value of $J(\theta_0, \theta_1)$ will reduce most. Again, we will advance by a tiny step in that direction.
- On repeating this many more times, we will reach at point P_6 which is our local minimum, i.e., the minimum cost.

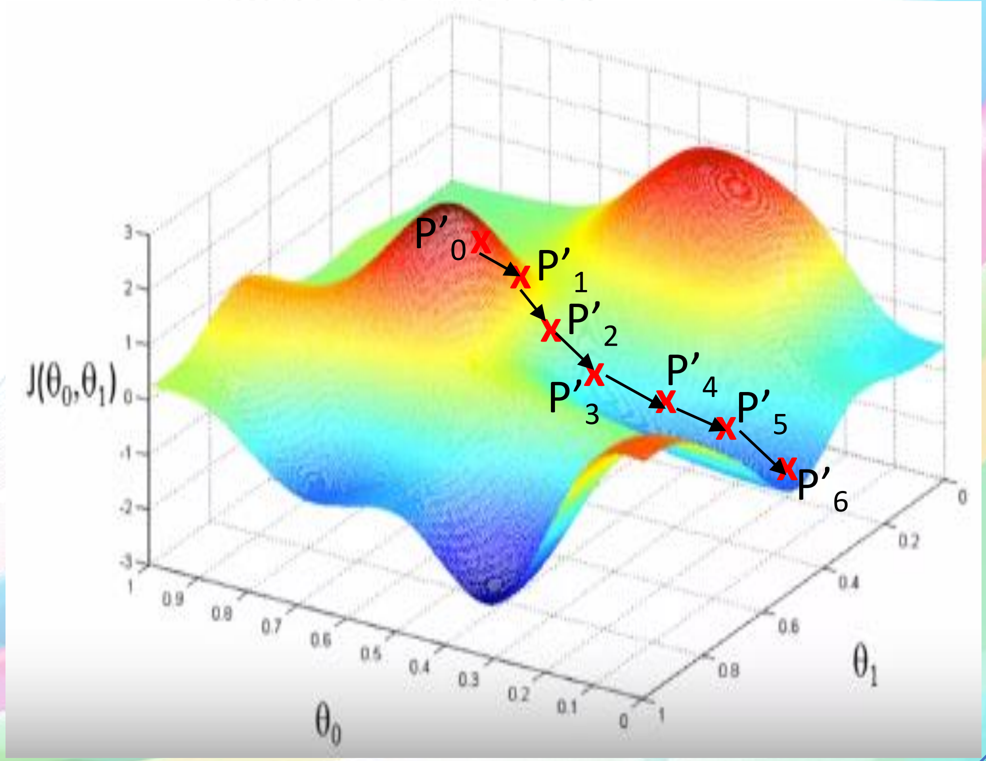


Parameter Learning

Gradient Descent

What Gradient Descent does?

- Suppose, this time we have initialised θ_0 & θ_1 with some other random value, due to which you are at position P'_0 in the shown graph.
- Now, Gradient Descent will help us to find out the direction in which the value of our cost function $J(\theta_0, \theta_1)$ will reduce most by moving a tiny step (or baby step ☺) in that direction.
- Consequently, we will reach at P'_1. Again, Gradient Descent will help us to find a direction when the value of $J(\theta_0, \theta_1)$ will reduce most. Again, we will advance by a tiny step in that direction.
- On repeating this several times, we will reach at point P'_6 which is our local minimum, i.e., the minimum cost, which is different than last time. This is a property of Gradient Descent.



Parameter Learning

Gradient Descent

- Following is the definition of Gradient Descent:

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(For j = 0 and j = 1)}$$

}

Following is the correct simultaneous update of above function:

$$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \qquad \text{(for j = 0)}$$

$$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \qquad \text{(for j = 1)}$$

$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$

Above update steps should be follow in the same order, otherwise change in order can raise a blunder error.

α: This is the learning rate. It controls the length of our step by which we will descend in a particular direction. More on this will come later.

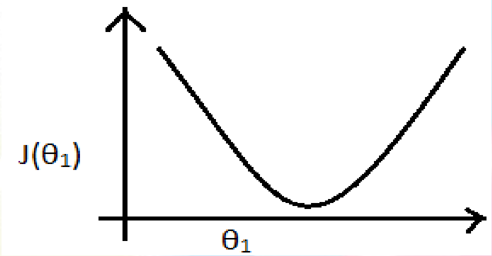
$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$: This is a derivative term that we will discuss later in more detail.

san.singhsanjay@gmail.com

Parameter Learning

Gradient Descent

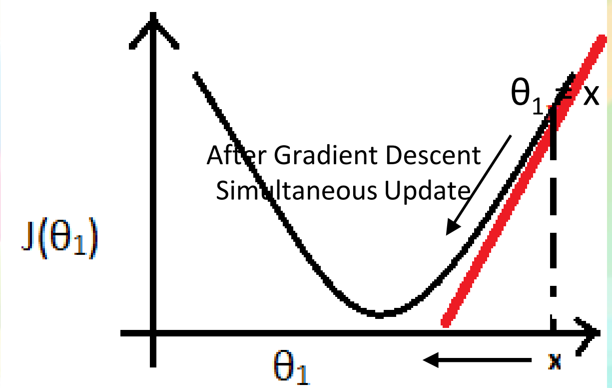
- Let us consider an example of one variable: θ_1
- Suppose, following is the graph of $J(\theta_1)$:



- According to Gradient Descent Simultaneous Update:

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

- As we know that $\frac{\partial}{\partial \theta_1} J(\theta_1)$ is the derivative term..
- The job of derivative term is to provide the slope of tangent at any θ_1 (say $\theta_1 = x$).
- Since, in the shown graph, the slope of line at $\theta_1 = x$, is positive, hence the value of overall Gradient Descent Simultaneous Update will decrease and take θ_1 towards the minimum.



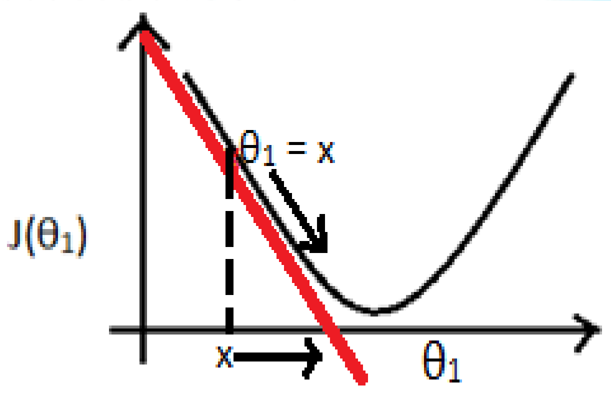
Parameter Learning

Gradient Descent

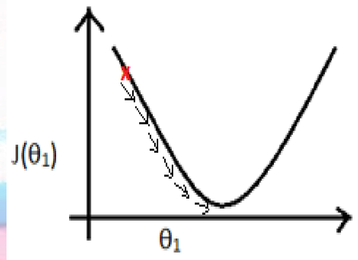
- According to Gradient Descent Simultaneous Update:

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

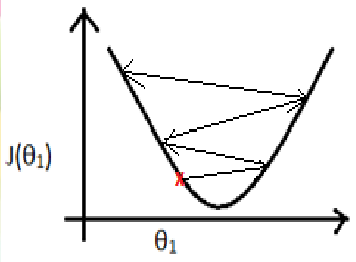
- As we know that $\frac{\partial}{\partial \theta_1} J(\theta_1)$ is the derivative term..
- The job of derivative term is to provide the slope of tangent at any θ_1 (say $\theta_1 = x$). This, time $\theta_1 = x$ is at left side of plot, as shown in the graph.
- Since, in the shown graph, the slope of line at $\theta_1 = x$, is negative, hence the value of overall Gradient Descent Simultaneous Update will increase and take θ_1 towards the minimum.



If learning rate (α) is too small, then gradient descent can be slow.



If learning rate (α) is too large, then gradient descent can overshoot the minimum. It may fail to converge or even diverge.



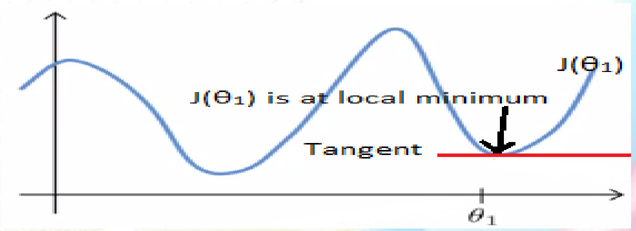
Week 1

Parameter Learning

Gradient Descent

When θ_1 is already at its local minimum, then the slope of tangent would be 0, that will make the derivative term zero, hence the value of θ_1 remain unchanged.

$$\theta_1 := \theta_1 - \alpha\, 0$$
$$\theta_1 := \theta_1$$

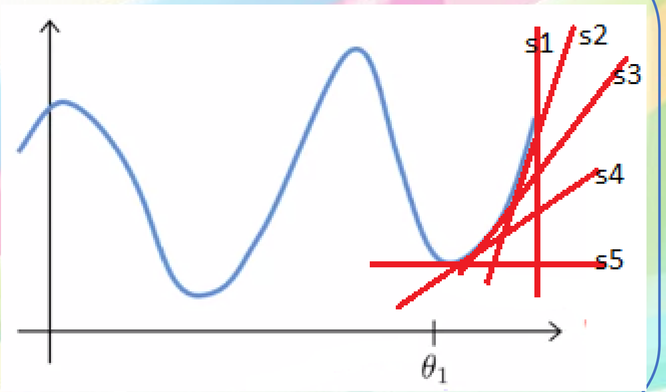


Gradient Descent can converge to a local minimum, even with the learning rate (α) fixed.

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

As θ_1 reaches close to its local minimum, the slope of tangent will reduce and make the overall product of learning rate (α) and the derivative term smaller and smaller, resulting in smaller change in θ_1. Thus, no need to decrease α over time.

Thus, the slope of tangent is in this order: s1 > s2 > s3 > s4 > s5.



Parameter Learning

Gradient Descent for Linear Regression

- So far, we have covered following two things:

Gradient Descent Algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(For j = 0 and j = 1)

}

Linear Regression Model

Hypothesis: $h(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h(xi) - y^i)^2$

- Now, we will apply Gradient Descent algorithm to minimize the cost function of Linear Regression model, that is:

$$\min_{(\theta_0, \theta_1)} J(\theta_0, \theta_1)$$

- The partial derivative term of Gradient Descent plays a key role in achieving the above objective, that is, minimising cost function.
- In the next slide, we will see the maths of this.

Parameter Learning

Gradient Descent for Linear Regression

- Following is the maths required to minimise the cost function:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - yi)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^i - yi)^2$$

Now, the partial derivative for j = 0 and j = 1 is:

For j = 0: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi)$

For j = 1: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi) \, . \, xi$

By learning the partial derivative, one can easily solve and find the above equations.

Thus, the simultaneous update of Gradient Descent will look like:

$$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi)$$

$$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi) \, . \, xi$$

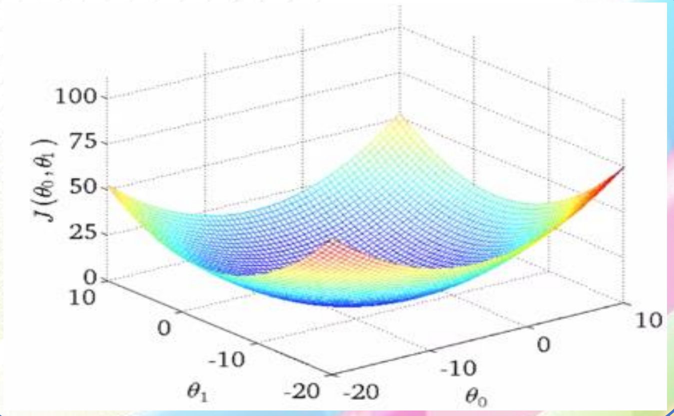
$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$

Parameter Learning

Gradient Descent for Linear Regression

- The plot of cost function of Linear Regression is actually a bowl shaped function (see the following image), formally called as “Convex Function”.



- One property of Convex functions is that they always have one local minima which coincides with the global minima.

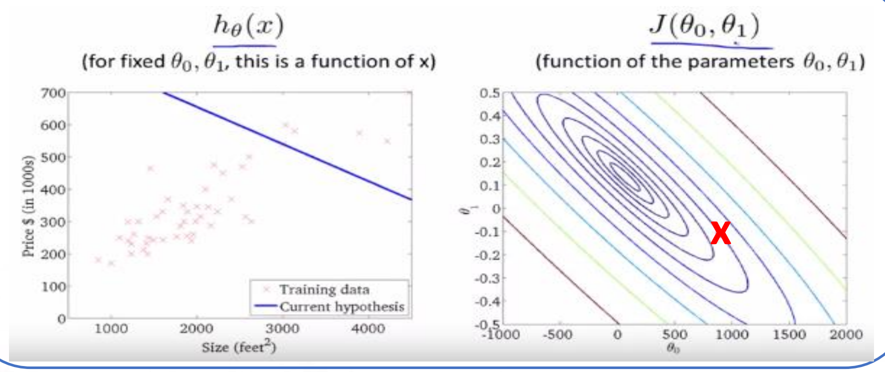
Week 1

Parameter Learning

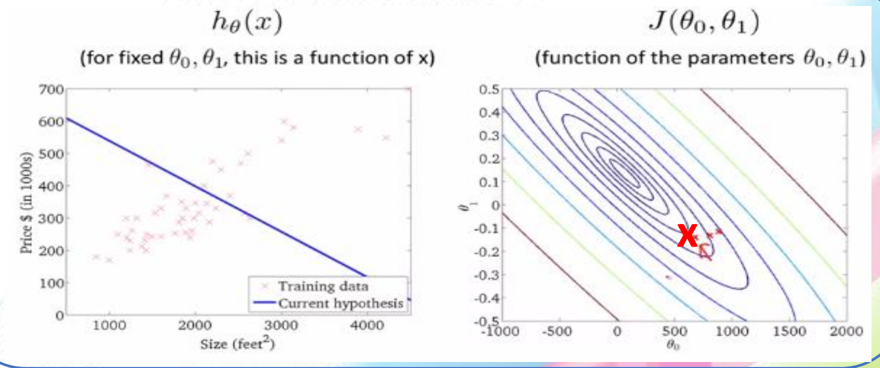
Gradient Descent for Linear Regression

- Following are some of the iterations with different values θ_0 and θ_1. Generally, we start with $\theta_0 = 0$ and $\theta_1 = 0$, but here we have started with some random values.

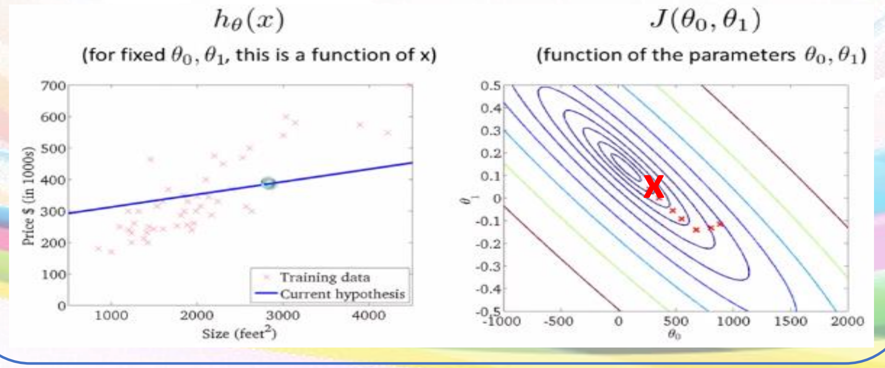
1



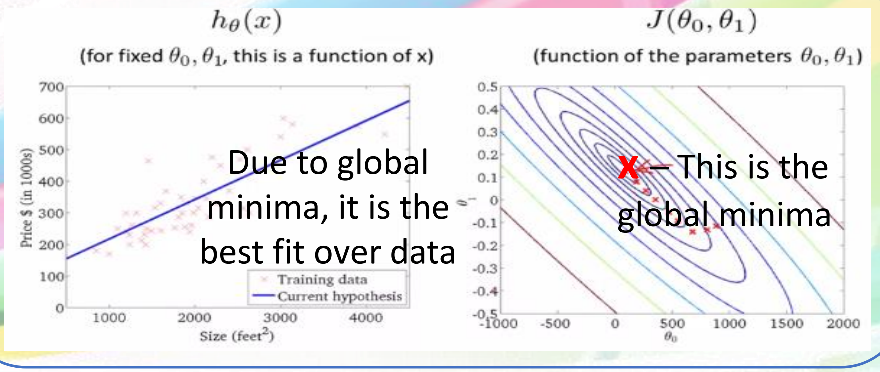
2



3



4



Parameter Learning

Gradient Descent for Linear Regression

- The Gradient Descent algorithm that we have seen so far is specifically called as Batch Gradient Descent.
- Batch: Each step of Gradient Descent uses all the training instances.
- Quiz:

  Which of the following are true statements? Select all that apply.

  1. To make Gradient Descent converge, we must slowly decrease α over time.
  2. Gradient Descent is guaranteed to find the global minimum for any function (θ_0, θ_1).
  3. Gradient Descent can converge even if α is kept fixed, but α cannot be too large, or else it may get fail to converge.
  4. For the specific choice of cost function $J(\theta_0, \theta_1)$ used in linear regression, there are no local optima (other than global optimum).

  Ans: 3 & 4
- Gradient Descent is an iterative algorithm as it computes the value of its parameters (θ_0, θ_1) again and again, and converged. However, in the Advanced Linear Algebra, there are methods to do the same thing, i.e., finding minimum of parameters (θ_0, θ_1), in one step. We will look at those methods later.
- That Advanced Linear Algebra method is Normal Linear Equation method, but Gradient Descent is better than that as Gradient Descent scales better on huge data set than Normal Linear Equation method.

Week 1

Linear Algebra Review

- Following are the topics covered under this heading:
  1. Matrices and Vectors
  2. Addition and Scalar Multiplication
  3. Matrix Vector Multiplication
  4. Matrix Matrix Multiplication
  5. Matrix Multiplication Properties
  6. Inverse and Transpose

Week 2

Multivariate Linear Regression

Multiple Features

- Earlier, we considered Linear Regression with single feature. In that case, data was as shown in figure:
- In this case, h(x) is the hypothesis function.

Size (feet²) x	Price ($1000) y
2104	460
1416	232
1534	315
852	178
…	…

$$h_\theta(x) = \theta_0 + \theta_1 x$$

- Here, we will consider Linear Regression with multiple features.
- In this case, {size, number of bedrooms, number of floors, age of home} are the features and {price} is the target variable.
- We will use {x_1, x_2, x_3, x_4} to represent features and {y} to represent the target variable, “price”.

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Multivariate Linear Regression

Multiple Features

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

- Notations:
  - n: Number of features, here it is 4.
  - x^i: All input features of i^{th} training example
  - x^i_j: Value of feature j in i^{th} training example
  - m: Total number of training examples
- Thus, $x^{=}$ = [1416, 3, 2, 40] and $x^2_3 = 2$
- Earlier, in case of single features, our hypothesis function was:

$$h(x) = \theta_0 + \theta_1 x$$

- However, now in case of multiple features, our hypothesis function will be (for n = 4):

$$h(x) = \theta 0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

- This can be generalised as:

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = \theta_0 + \sum_{k=1}^{n} \theta_k x_k$$

- Another form of writing the above equation:

Suppose, $x = [x_0, x_1, x_2, ..., x_n]$, here $x_0 = 1$ (always) and $\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ . \\ . \\ \theta_n \end{pmatrix}$, $\theta^T = [\theta_0, \theta_1, ..., \theta_n]$. Thus, $h(x) = \theta^T x$. This is called as **Multivariate Linear Regression**.

Multivariate Linear Regression

Gradient Descent for Multiple Features

- Now, we have following things:
  - Hypothesis function: $h(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$, where $x_0 = 1$.
  - Parameters: $\{\theta_0, \theta_1, \theta_2, \dots, \theta_n\}$, let us call it simply θ.
  - Cost function: $J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - yi)^2$
- Now, following is the algorithm of Gradient Descent for multiple features:

$$\theta_j := \theta j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi) x^i{}_j$$

Simultaneously update θ_j for j = 0, 1, 2, …, n.

Here, α is the learning rate.

Above term, i.e., $\frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi) x^i{}_j$, is actually the partial derivative of cost function J(θ), i.e., $\frac{\partial}{\partial \theta_j} J(\theta)$.

Compare it with the Gradient Descent equation for single variable and comprehend it properly.

Thus, if we have two features, then:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi) x^i{}_0$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi) x^i{}_1$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - yi) x^i{}_2$$

Where, $x_0 = 1$.

Week 2

Multivariate Linear Regression

Gradient Descent in Practice I – Feature Scaling

- Here, we will discuss a few practical tricks to make Gradient Descent work well.
- One of such tricks is Feature Scaling.
- Feature Scaling: make sure that features are on a similar scale.

For example:

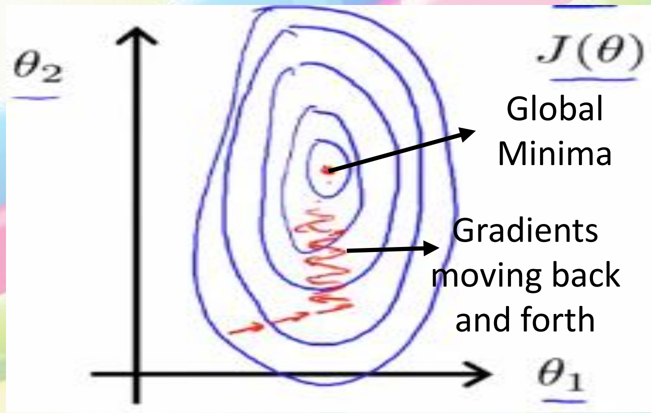
x_1: size of apartment (0 – 2000 sq. feet)

x_2: number of bedrooms (1 – 5)

The cost function, $J(\theta)$, for above features will have three parameters, $J(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

Let us ignore θ_0 for a while and see the plot of θ_1 and θ_2:

- In this plot, θ_1 is on x-axis and θ_2 is on y-axis.
- The plot is of cost function $J(\theta)$ (i.e., $\theta_1 x_1 + \theta_2 x_2$, θ_0 ignored).
- Because of wide range of x_1 (i.e., 0 – 2000) and very narrow range of x_2 (i.e., 1 – 5), the contours of $J(\theta)$ will be very skewed (i.e., tall and skinny) and elliptical in shape.
- Due to this, the Gradient Descent algorithm will take a very long time to reach at global minima at the centre as the gradient will be changing back and forth (see zig-zag red arrows).



Multivariate Linear Regression

Gradient Descent in Practice I – Feature Scaling

- The previous issue of taking a very long time for convergence can be resolved by scaling the features.
- This can be done as:

  Our actual features:

  x_1: size (0 – 2000 square feet)

  x_2: number of bedrooms (1 – 5)

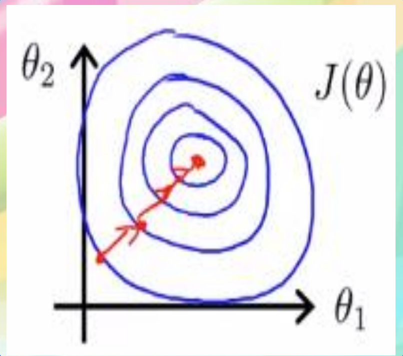
  Now, scaled x_1 and x_2:

  $$x_1 = \frac{size\ (feet^2)}{2000} \qquad \text{and} \qquad x_2 = \frac{number\ of\ bedrooms}{5}$$

  The above mathematical operation will result into: $(0 \leq x_1 \leq 1)$ & $(0 \leq x_2 \leq 1)$.

  In this case, the contours of cost function $J(\theta)$, i.e., $\theta_1 x_1 + \theta_2 x_2$ (ignore θ_0), will look like this:

  - The contours of $J(\theta)$ will become less skewed (i.e., tall and skinny) or may be circular.
  - Due to this, the Gradient Descent will perform substantially better.
  - And Gradient Descent algorithm would be able to reach at the centre, i.e., Global minima, quite earlier. That is, the convergence will be much faster.



Multivariate Linear Regression

Gradient Descent in Practice I – Feature Scaling

- Generally, when we do feature scaling, we often want our features to be in the range of -1 and +1:

$$-1 \leq x_i \leq +1$$

- The number -1 and +1 are not important. This can be possible that the range of a feature after scaling is 0 &1 or 0 & +3 or -2 & +0.5, etc. Since these ranges are quite close to -1 & +1 range, hence they are fine.
- However, if the ranges are substantially higher or lower than -1 & +1, such as:

$$-100 \leq x_i \leq +100 \quad \text{or} \quad -0.0001 \leq x_i \leq +0.0001$$

  Then, this is called as poorly scaled feature.
- Different people have different thumb rule for the accepted range after feature scaling. One such rule that has been accept by Andrew Ng is around -3 to +3 or -0.333 to +0.333.
- Sometime, people also take another approach of scaling, called as Mean Normalization.
- Mean Normalization:
  - Replace x_i with $(x_i – \mu_i)$, where μ_i is the average value of x_i, to make features have approximately zero mean (a data with mean 0 as standard normal distribution has mean 0 and variance 1).
  - Do not do it for x_0, as $x_0 = 1$.
  - For instance, when use both Mean Normalization and Feature Scaling, i.e., $(x_i – \mu_i) / (\max(x_i) – \min(x_i))$:

$$x_1 = \frac{size - 1000}{2000} \quad \text{and} \quad x_2 = \frac{\#bedrooms - 2}{5}$$

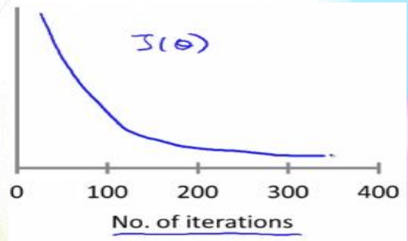
  This will result in: $-0.5 \leq x_1 \leq +0.5$ and $-0.5 \leq x_2 \leq +0.5$

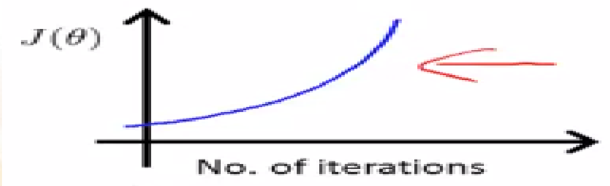
Week 2

Multivariate Linear Regression

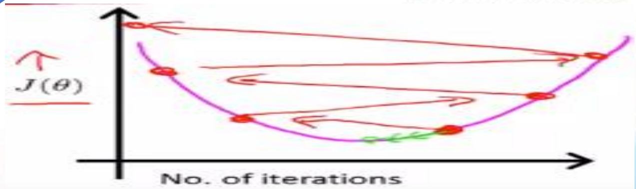
Gradient Descent in Practice II – Learning Rate

- In the corresponding graph, number of training iterations is on x-axis and the value of cost function J(θ) is on y-axis.
- To make sure that the gradient descent is working fine, we should plot the values of J(θ).
- If the plot is declining with each iteration of training, then it means that the gradient descent is working fine and cost is reducing.
- Thus, J(θ) should decrease after each iteration.





- If this plot of J(θ), cost is increasing.
- This can be resolved by further reducing learning rate



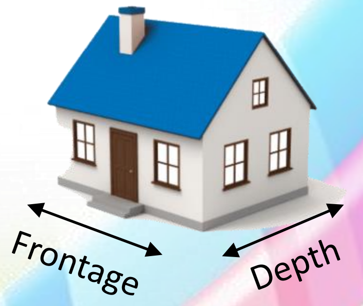
- Here, gradients are overshooting due to higher learning rate.
- Thus, reduce learning rate.

- If the plot of J(θ) is weird like above, then also reduce the learning rate.

- For sufficiently small learning rate (α), J(θ) should decrease on every iteration
- But, if learning rate (α) is too small, gradient descent can be extremely slow to converge.

Multivariate Linear Regression

Features and Polynomial Regression



- Suppose, we have two features for house pricing prediction problem:
  i. Frontage
  ii. Depth
- The hypothesis function $h(x_1, x_2)$ for this problem can be like this:

$$h(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

  Where, x_1: frontage and x_2: depth
- Above we have two features, but we can create a new feature by combining them.
  For instance:

$$\text{area}, x = x_1 * x_2 = \text{frontage x depth}$$

  Thus, our new hypothesis function h(x) would look like this:

$$h(x) = \theta_0 + \theta_1 x$$

- Sometimes, by defining new features, we may get a better model.

Multivariate Linear Regression

Features and Polynomial Regression

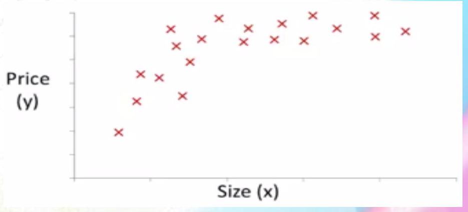
- In the given graph, we have size (i.e., area = frontage x depth) of house on x-axis.
- Price is on the y-axis.
- Apparently, it is clear from the plot that a linear regression cannot fit this.
- Thus, we have to use a polynomial regression.
- In order to perform that, suppose first we have chosen the quadratic model.
- Following are the equations of linear regression model and quadratic model:

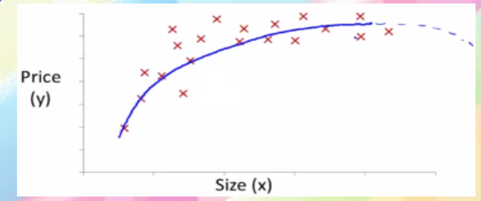
  Linear Regression: $h(x) = \theta_0 + \theta_1 x$

  Quadratic Regression: $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

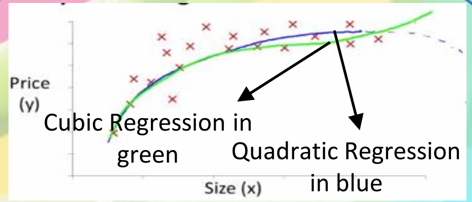
  Where, x: area
- The Quadratic Regression model will fit in the beginning, but later it will not.
- It is shown in the graph "Quadratic Regression" that the model will decline in later stages.
- However, we know that the prices will increase further in later stages.
- To rectify this, we can use a cubic function or cubic regression, as shown in the figure.
- Cubic regression: $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- To implement quadratic or cubic regression, create three variables in data set:

  x_1: x (i.e., area) x_2: x^2 (i.e., area^2) x_3: x^3 (i.e., area^3)
- Since ranges of x_1, x_2 and x_3 will get changed, hence do not forget to do feature scaling with their respective ranges.





Quadratic Regression



Cubic Regression

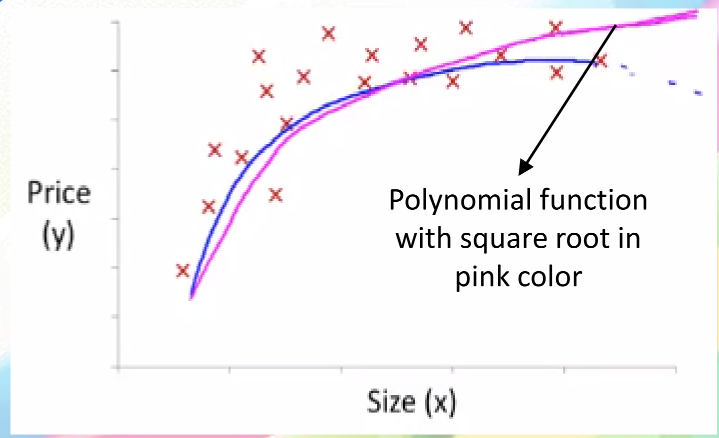
Multivariate Linear Regression

Features and Polynomial Regression

- Sometimes, another type of polynomial function can be more fruitful to fit the data.
- One such example is a polynomial function with a square root:

$$h(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x}$$

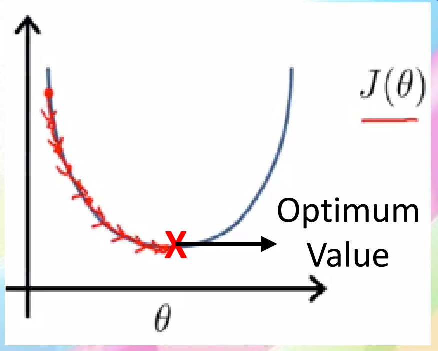
- From the graph, we can see that the plot is increasing slightly in the later stages.



Computing Parameters Analytically

Normal Equation

- Normal Equation gives us much better way to solve for the optimum value of the parameters of θ for some linear regression.
- So far we have seen that the Linear Regression make use of Gradient Descent that works in steps to figure out the optimum values of θs, as shown in the figure.
- However, Normal Equation works analytically to figure out the optimum values of θs in one go, instead of several steps.
- Now, suppose there is a cost function, $J(\theta) = a\,\theta^2 + b\,\theta + c$
- To minimize this cost function, J(θ), we have to take its partial derivative and equate it with zero in order to find the values of θs. Something like as shown below:



$$h(x_1, x_2, \ldots, x_n) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots. + \theta_n x_n$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - y^i)^2$$

$$\frac{\delta}{\delta\theta_j} J(\theta) = \ldots\ldots. = 0 \text{ (for every j)}$$

Solve for $\theta_0, \theta_1, \theta_2, \ldots., \theta_n$.

Computing Parameters Analytically

Normal Equation

- Unlike the partial derivative method to find the optimised values of θs in several steps to minimise the cost function J(θ), here we will make use of normal equations to achieve the same thing, but in one step.
- Suppose, we have the following data set with total rows, m = 4; and total features, n = 5 (including $x_0 = 1$):

x_0	Size (feet^2) X_1	No. of bedrooms x_2	No. of floors x_3	Age of home (yrs.) x_4	Price ($1,000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

- Create a matrix of features and target values:

$$X = \begin{pmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{pmatrix} \qquad y = \begin{pmatrix} 460 \\ 232 \\ 315 \\ 178 \end{pmatrix}$$

To get values of θs that minimise your cost function J(θ), just solve this: $\theta = (X^T X)^{-1} X^T y$

- In the method of Normal Equation, Feature Scaling is not required. Thus, it doesn't matter if one feature is in range of [0, 1] and another in range of [0, 1000] or [0, 10^{-5}].

Week 2

Computing Parameters Analytically

Gradient Descent vs Normal Equation

#	Gradient Descent	Normal Equation
1	[Cons] Need to choose learning rate (α)	[Pros] No need to choose learning rate (α)
2	[Cons] Needs many iterations to minimise cost function $J(\theta)$	[Pros] No need to iterate
3	[Pros] Works well even when number of features, n, is very large	[Cons] Since it has to compute $(X^T X)^{-1}$, hence it is extremely slow when number of features, n, is large.

- For smaller value of number of features, i.e., n, Normal Equation will run much faster than Gradient Descent.
- Now, how to figure out that for which value of n (i.e., number of features), Normal Equation will be extremely slow and it would be better to choose Gradient Descent. Following are some guidelines from Andrew Ng:
  In today's modern computers, we can use Normal Equation for value of n up to 10,000. Since the runtime complexity of n in case of Normal Equation becomes $O(n^3)$, hence for number of features greater than 10,000, it would be better to use Gradient Descent.

Computing Parameters Analytically

Normal Equation Non-invertibility

- We know that the Normal Equation has this formula to calculate the optimised values of θs:

$$\theta = (X^T X)^{-1} X^T y$$

Now, what if $(X^T X)$ is non-invertible, i.e., it is a singular matrix?

In Octave, there are two functions to calculate the inverse of any matrix:

i. pinv: sudo-inverse function. This function will provide the values of θ even if $(X^T X)$ is non-invertible.
ii. inv: inverse function. This function will not work if $(X^T X)$ is non-invertible.

Following can be the cause of non-invertibility of $(X^T X)$:

i. Redundant features (linearly dependent): Suppose you have two features which are linearly related to each other, such as, x_1: size in feet^2 and x_2: size in meter^2. Here, $x_1 = (3.28)^2 x_2$ is the relation between x_1 and x_2.

   In such situations, delete one such linearly dependent feature to remove the redundancy.
ii. Too many features: In this case, the number of features (i.e., n) would be much larger than the number of instances (i.e., m). That is, $n \geq m$.

   Two solutions for this case:

   a. Delete some features that are clearly less / not important.
   b. Use regularization.

Week 2

Computing Parameters Analytically

Octave / Matlab tutorial

Skipped

Topics Covered

1. Basic Operations
2. Moving Data Around
3. Computing on Data
4. Plotting Data
5. Control Statements – for, while, if
6. Vectorization

Week

Template

- Template

Week

Template

- Template

Thank You